SFA Modernization Partner

United States Department of Education

Student Financial Assistance

SFA Portal Strategy
Design Specification
2.p

Task Order #48

November 2, 2001

Table of Contents

1 Architecture Overview

1.1 Introduction

This document explains the design for the SFA Portal Strategy Task Order #48. The intended audience is the project managers and developers for Task Order #48. The pre-requisites for understanding this document are:

- Understanding of the control flow and function of the current version of the SFA Portal Web site.
- Understanding of Web Sphere technology in general.
- Understanding of Struts technology in general.
- Understanding of ***SFA Portal Strategy Portal Software Architecture*** (#48.1.1)

1.2 The Role of Apache Struts

The J2EE APIs provide a flexible set of facilities for building Web-based applications. However, faced with the wide set of design alternatives available, the problem is that developers are too often faced with "reinventing the wheel" each time they begin building a new web-based application. Such applications become difficult to develop, as well as extend and maintain.

There are a number of common problems that must be solved in every project [ref 1.]:

- Mapping HTTP parameters to JavaBeans—One of the most common tasks facing servlet programmers is to map a set of HTTP parameters (from the command line or from the POST of an HTML form) to a JavaBean for manipulation. This can be done using the <jsp:useBean>and <jsp:setProperty>tags, but this arrangement is cumbersome as it requires POSTing to a JSP, something that is not encouraged in a Model-II Model View Controller architecture.
- Validation—There is no standard way in servlet/JSP programming to validate that an HTML form is filled in correctly. This leaves every servlet programmer to develop his own validation procedures, or not, as is too often the case.
- Error display—There is no standard way to display error messages in a JSP page or generate error messages in a servlet.
- Message internationalization—Even when developers strive to keep as much of the HTML as possible in JSPs, there are often hidden obstacles to internationalization spread throughout servlet and model code in the form of short error or informational messages. While it is possible to introduce internationalization with the use of Java resource managers, this is rarely done due to the complexity of adding these references.
- Hard coded JSP URIs—One of the more insidious problems in a servlet architecture is that the URIs of the JSP pages are usually coded directly into the code of the calling servlet in the form of a static string reference used in the ServletContext.getRequestDispatcher() method. This means that it is impossible to reorganize the JSPs in a Web site, or even change their names, without updating Java code in the servlets.

Apache Struts addresses these and other issues by providing an framework that isolates and encapsulates each of the model, view and control components of a Web application as described by the following diagram.

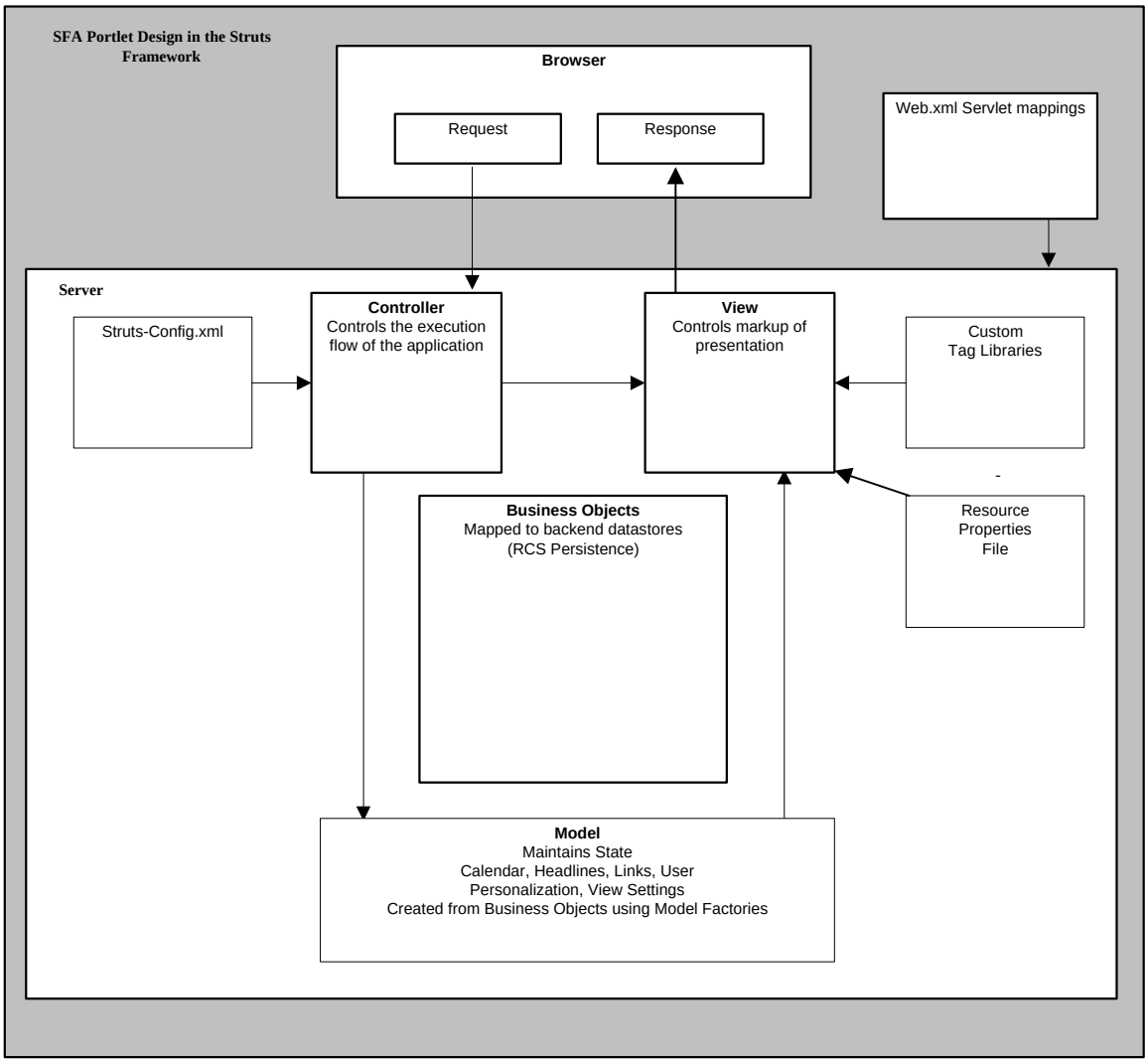


Figure 1 SFA Portlet Design in the Struts Framework

Struts, part of the Jakarta Project sponsored by the Apache Software Foundation, is an open source framework used in building Web applications with Java Servlet and Java Servlet Pages (JSP) technology using the model-view-controller (MVC) design paradigm. It provides a standard set of classes and interfaces and includes the following areas of functionality:

- A controller servlet that dispatches requests to appropriate Action classes provided by the application developer.
- JSP Custom Tag libraries, and associated support in the controller servlet, that assists developers in creating interactive form-based applications.
- Utility classes to support XML parsing, automatic population of JavaBeans properties based on the Java reflection APIs and internationalization of prompt and messages.

1.3 Portlet Architecture

Each portlet in a horizontal portal is generally a separate MVC application, loosely linked to the overall portal by ancillary requirements, such as security and visual integration (common look and feel). For productivity purposes it is desirable to partition the work so that a team of developers can work on portlets independently. Key to this is to have a framework available that provides a common set of MVC facilities, including:

- A navigation framework, which standardizes the mechanism used to route request processors and the target of those requests (JSP or HTML). In Struts the request processors are termed Actions.
- JSP Tag Libraries, which simplify the development of the portlet JSPs, since they reduce the amount of JSP scripting that needs to be provided. Struts provides a number of Custom Tag libraries. For example, the tag library eases specification of HTML forms by providing a set of message display facilities, buttons, and other forms-based visual controls. Portlet developers may further extend the set of tags to provide enhanced functionality as well as common look and feel between portlets.
- A simple means of binding portlet JSP parameters to Actions by means of some Java Bean having a set of properties, each one corresponding to an HTML form or query parameter. In Struts the parameter-binding object is termed a Form. In addition, Struts provides a standardized validation mechanism for HTML form parameters, very useful in forms intensive portlets.

Using Struts, the portlet developer's role is thus greatly simplified to providing Struts Actions, Forms, JSPs and JSP tags and integrating them by developing some additional Java Beans. See **Figure 2 Portlet Beans**.

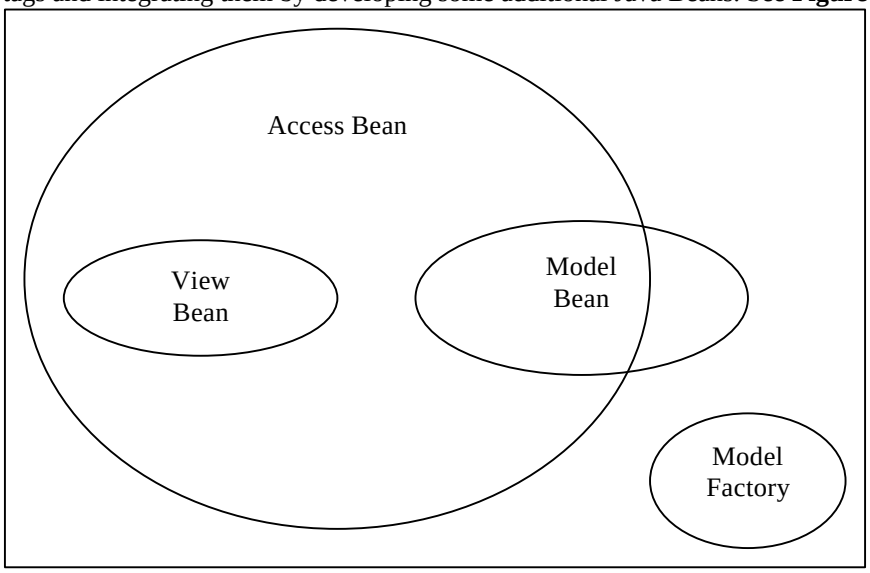


Figure 2 Portlet Beans

- Access Beans - In the web-based MVC paradigm, Struts Actions pass information to JSPs by placing Java Beans in the HttpRequest or Session object. The JSP retrieves these beans and has access to the required information in the bean properties. These beans are termed access beans since they allow JSPs access to information retrieved from the middle-ware. Note that these beans are passive, in the sense that they do not themselves interact with middle-ware, but generally consist of a set of properties, holding data only - in accordance with the MVC paradigm, all access to middle-ware is done in the Action object, not the JSP. Access beans usually contain properties or arrays of properties of type java.lang.String, or single or array references to other access beans, sometimes forming graphs of interconnected beans.
- View Beans - These are special kinds of access beans that are directly responsible for rendering HTML to the browser. They are used to emit a set of complex HTML such as the calendar tabular view or calendar header and can be thought of as visual 'widgets'. View beans have a naming convention, ending with 'View', e.g. SFACalendarView.
- Model Beans - These form the 'Model' part of the MVC paradigm. They generally contain business information of some kind, such as array collections of other beans holding dates and HTML links, and user information. This information often originates from some back-end data source such as a relational, XML or LDAP datastore. Some model beans can also be used as access beans, in that they may be passed to JSPs for presentation purposes. Others have methods that are used for computation purposes in the Action object. Model Beans have a RCS naming convention, ending with 'BO', e.g. SFACalendarBO.
- Model Factories - Model beans come into existence in different ways; some via 'Factory' classes, and others using constructors. Most of the time model beans contain information retrieved from back-end data stores such

as relational or XML data stores, or LDAP. The Bridge Pattern is used to separate the knowledge of the retrieval mechanism from the model bean itself. All retrieval of model data is done in a Model Factory class, using 'find' or 'create' methods. Model bean factories have a naming convention, ending with 'Factory', e.g. SFACalendarBOFactory. See **Integration with Backend Data** Sources for additional discussion of Model beans and Factories, and an example of their use.

To summarize, Struts implements the MVC presentation layer with JSPs, Form objects, Action objects and Access beans. Access beans used in the JSPs do not interact with middle ware. All such interaction is done in the Action object, most often via Factory objects.

The impact of restricting the kinds of beans used in the JSP to access beans is that the initial starting point for a portlet should be from an Action object, not a JSP, since in many cases (e.g. calendar), the initial view requires retrieved data to be displayed.

View beans are directly responsible for rendering to the browser. In many cases, they work in the following way. They have methods for setting internal state – these methods are invoked within the Action object that works together with the middle ware access beans to retrieve back-end data. The view beans are then forwarded to the appropriate JSP in the HttpRequest object. The view beans have one or more 'emit' methods to emit HTML to the Action object's output stream dependant on the state that has been set. We will walk though an example later on of the kind of HTML that needs to be emitted.

In most cases all the portlet JSP needs to do is to call the appropriate emit method in the appropriate location in the JSP. These calls are done using set of custom JSP tags, making writing the portlet JSPs extremely simple.

Other access beans may be kept in the HttpSession if necessary. One of these is discussed, as an example later, the SFACalendar bean, which needs to maintain state relating to the current date or date last-selected in the Calendar Portlet.

The SFA Portal site is composed of a horizontal portal, implemented as a frameset, linking to a set of individual vertical portlet action objects.

The frameset references the action object in the following fashion. Example for the Calendar Portlet is given below.

```
<FRAME src="/sfaportal/processCalendar.do?submit=+">
```

This is analogous to invoking a servlet from the frameset.

The Struts Action Framework transfers control to the ProcessCalendarAction object, passing query parameters via the CalendarForm object.

Each of the action objects when invoked in default state will forward to the associated default portlet JSP, which will render the initial state of each portlet page. See table below.

Table 1 Default action objects and JSPs

Portlet	**Default action object**	**Default JSP**
Horizontal Portal	DisplayHeaderAction	header.jsp
Calendar	ProcessCalendarAction	calendar.jsp
Login	DisplayLogonAction	logon.jsp
Registration	DisplayLogonAction	registration.jsp
Personalization	DisplayLogonAction	personalize.jsp
Links	DisplayLinksAction	links.jsp
Headlines	DisplayHeadlineAction	headlines.jsp
Search	DisplaySearchAction	search.html

All the portlet JSPs are written using a combination of:

- HTML
- Struts tags
- SFA custom tags.
- JSP scriptlet code. However there is very little, if any, JSP scriptlet code in most JSP pages.

1.4 Navigation Schematics

The set of Actions and Forms for the horizontal portal and each portlet contained within, and the way they relate to the flow of control to/from the JSPs are shown schematically in the figures below. (These are further discussed in Section 2 Portlet Detailed Designs where the tags and view beans are also described.)

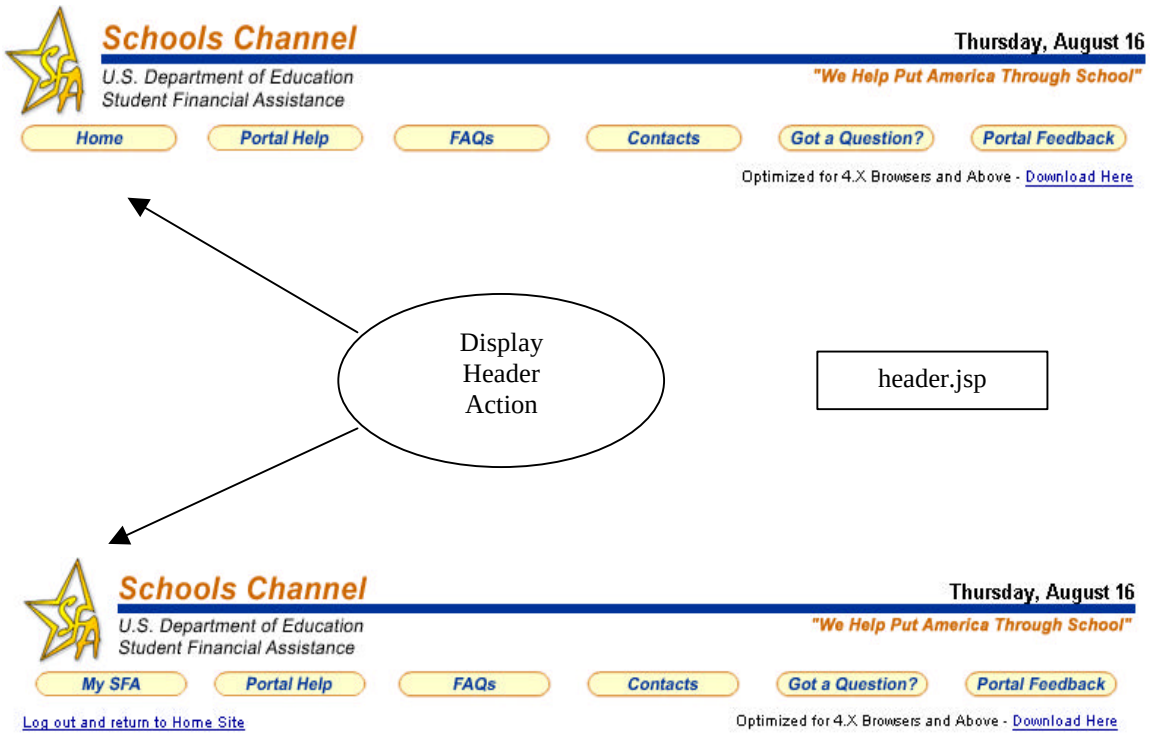


Figure 3 Portal Shared Header Navigation Schematic

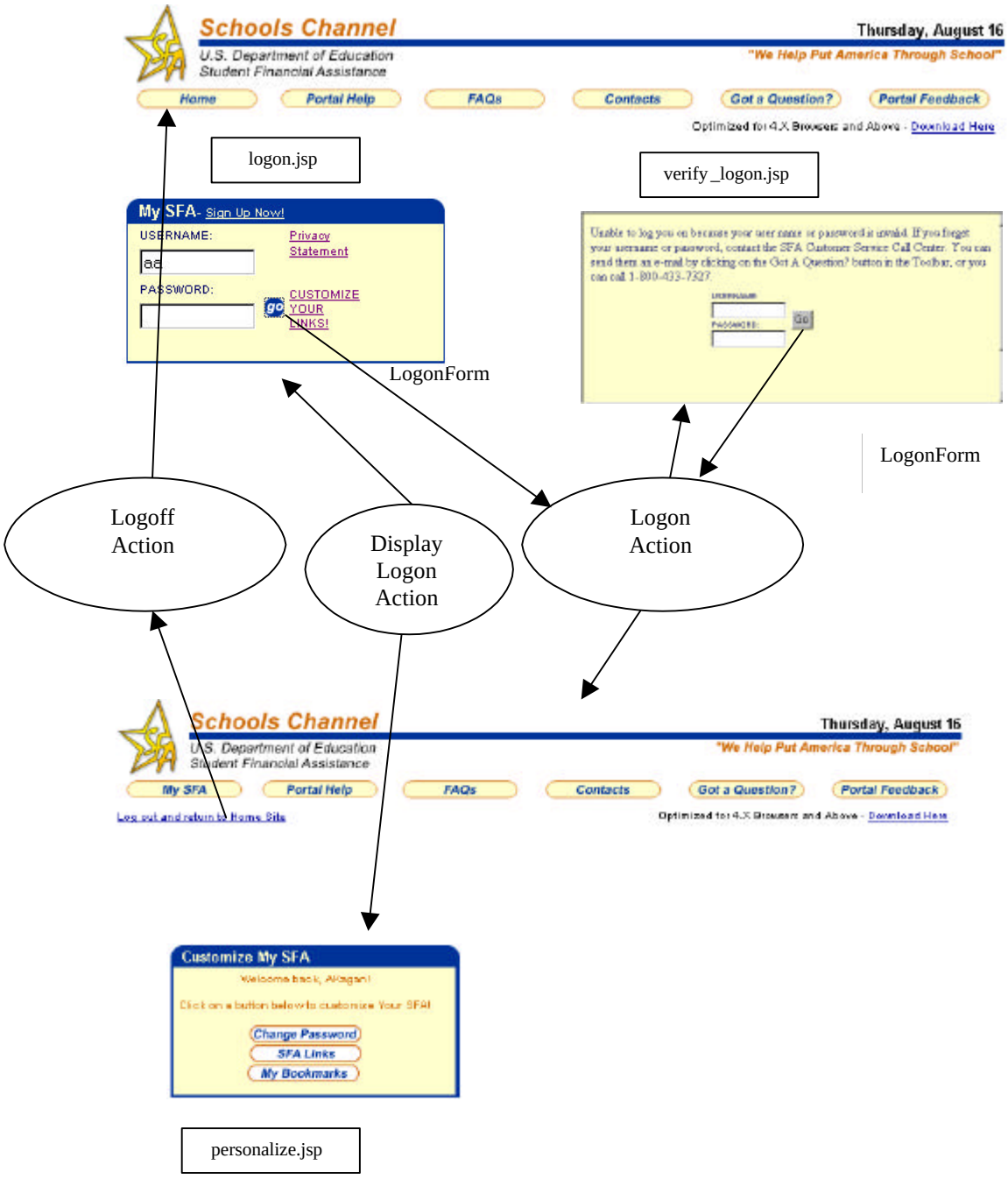


Figure 4 Login Portlet Navigation Schematic

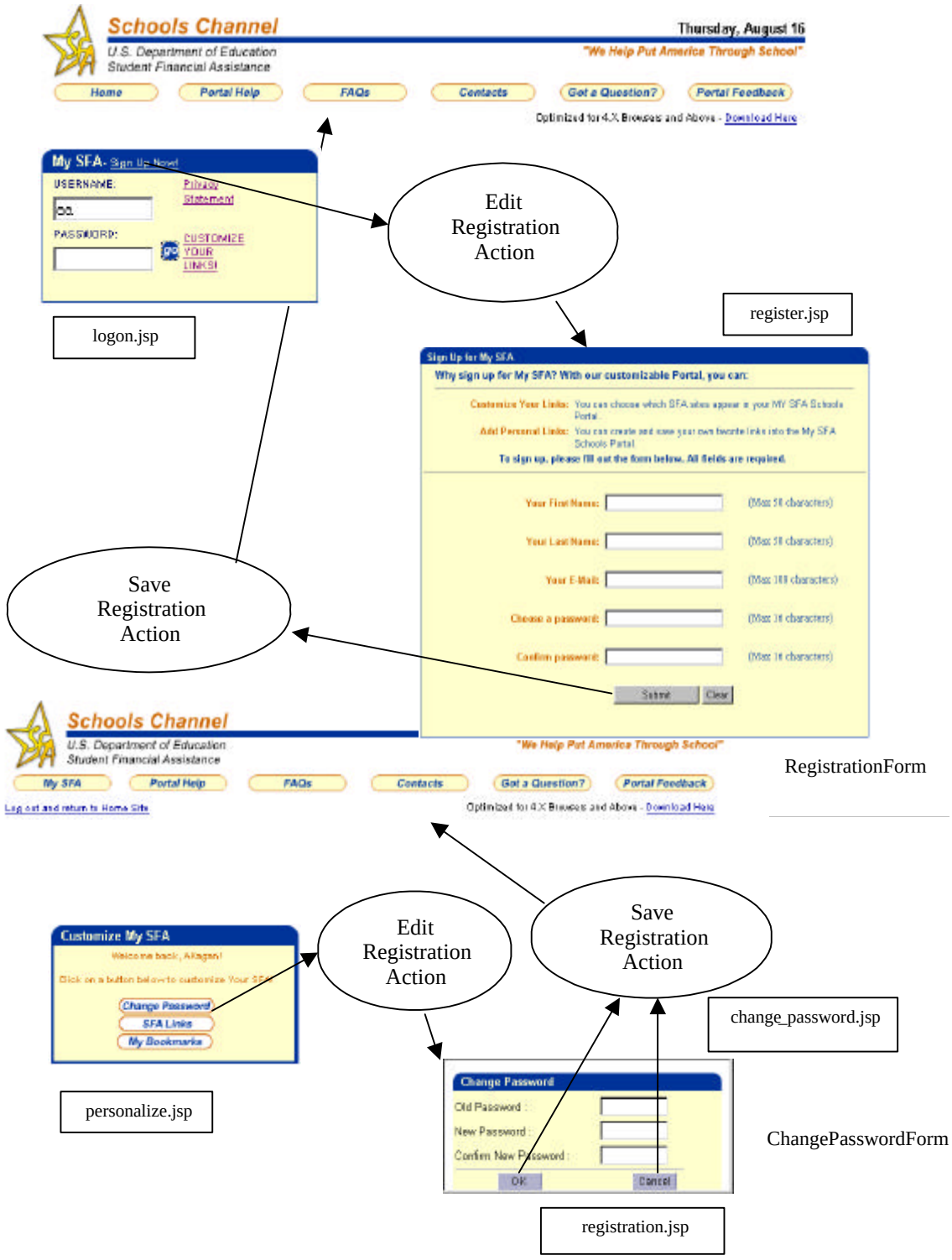


Figure 5 Registration Portlet Navigation Schematic

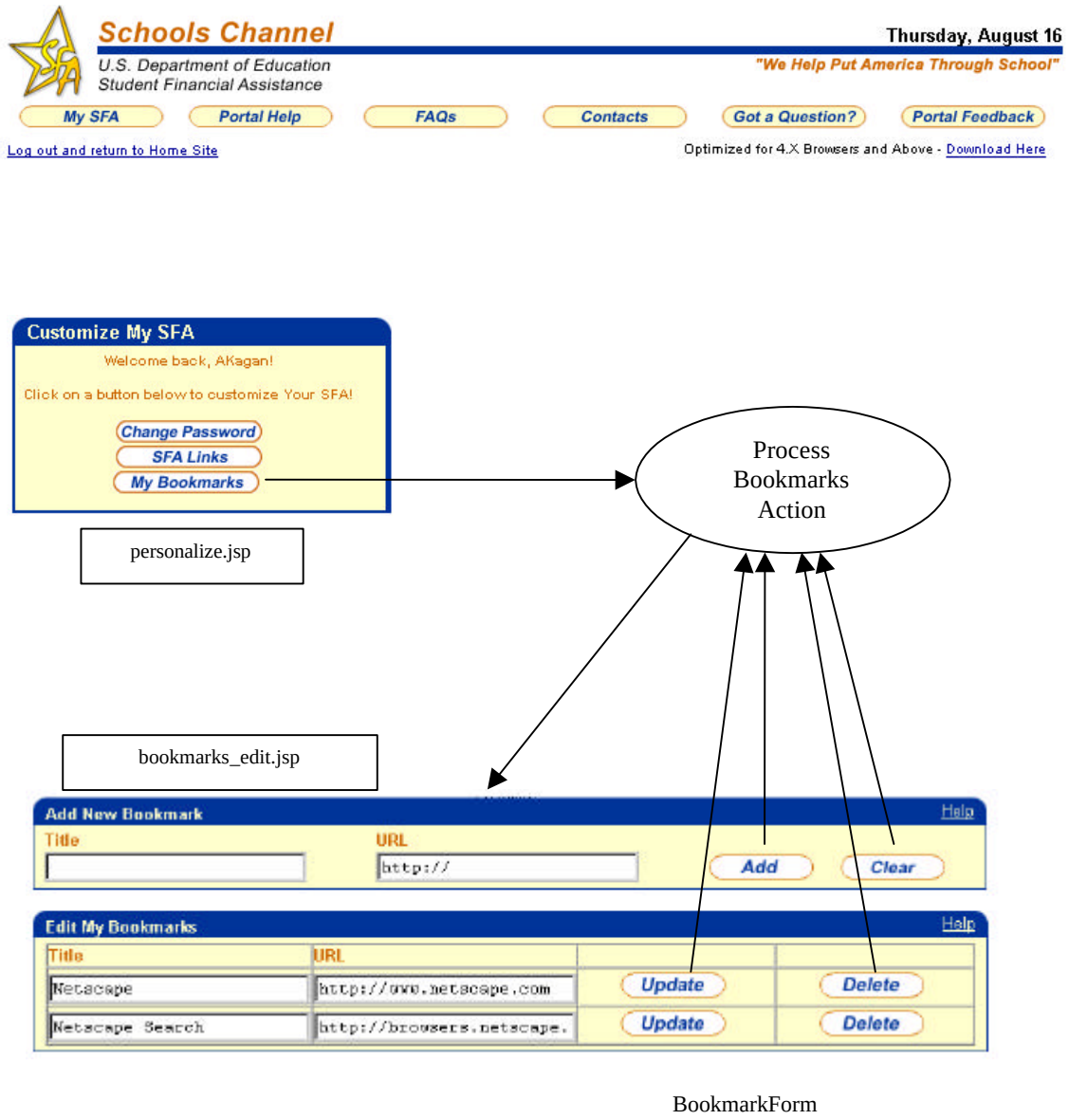


Figure 6 Personalization (Bookmarks) Portlet Navigation Schematic

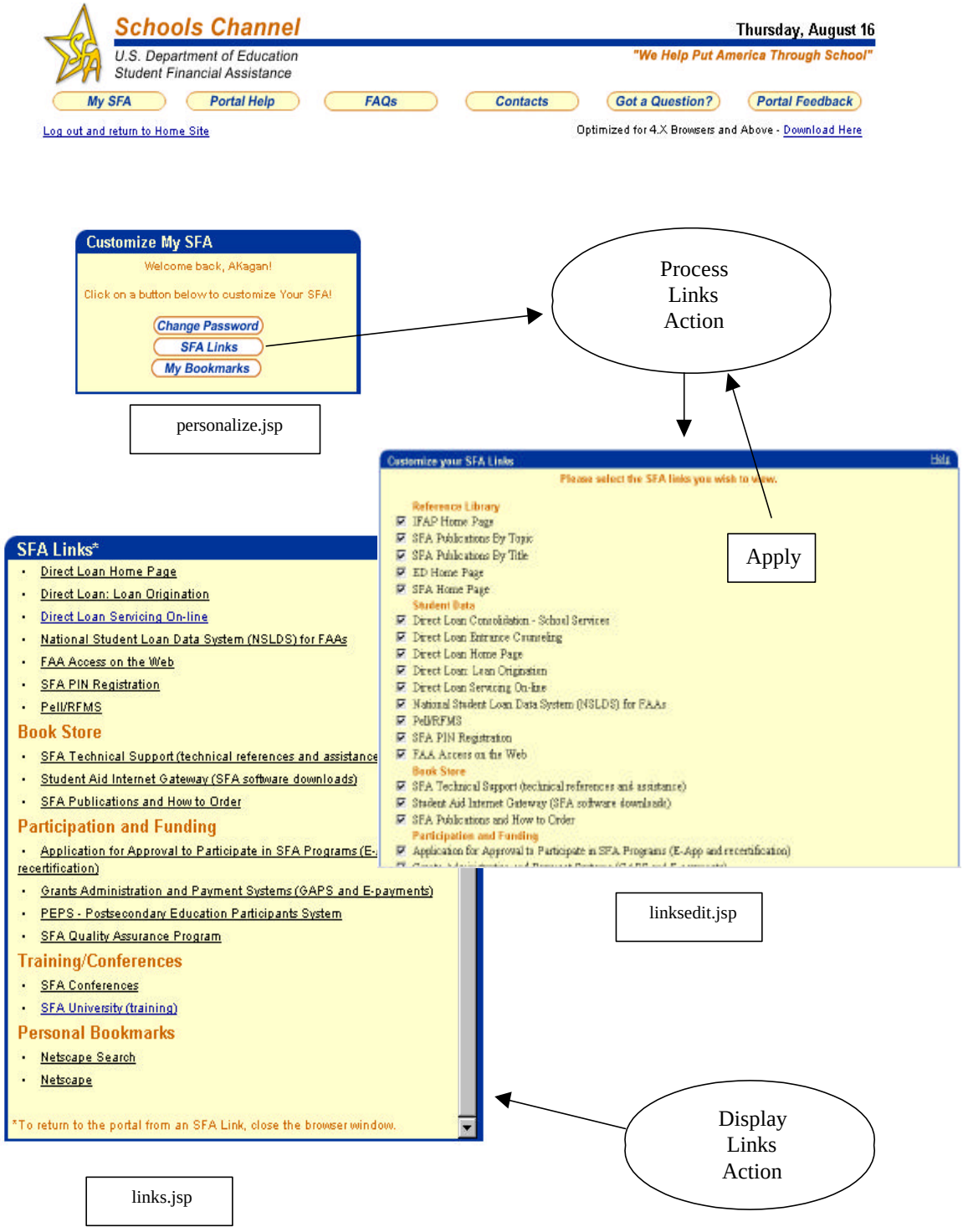


Figure 7 Personalization (Links) Portlet Navigation Schematic

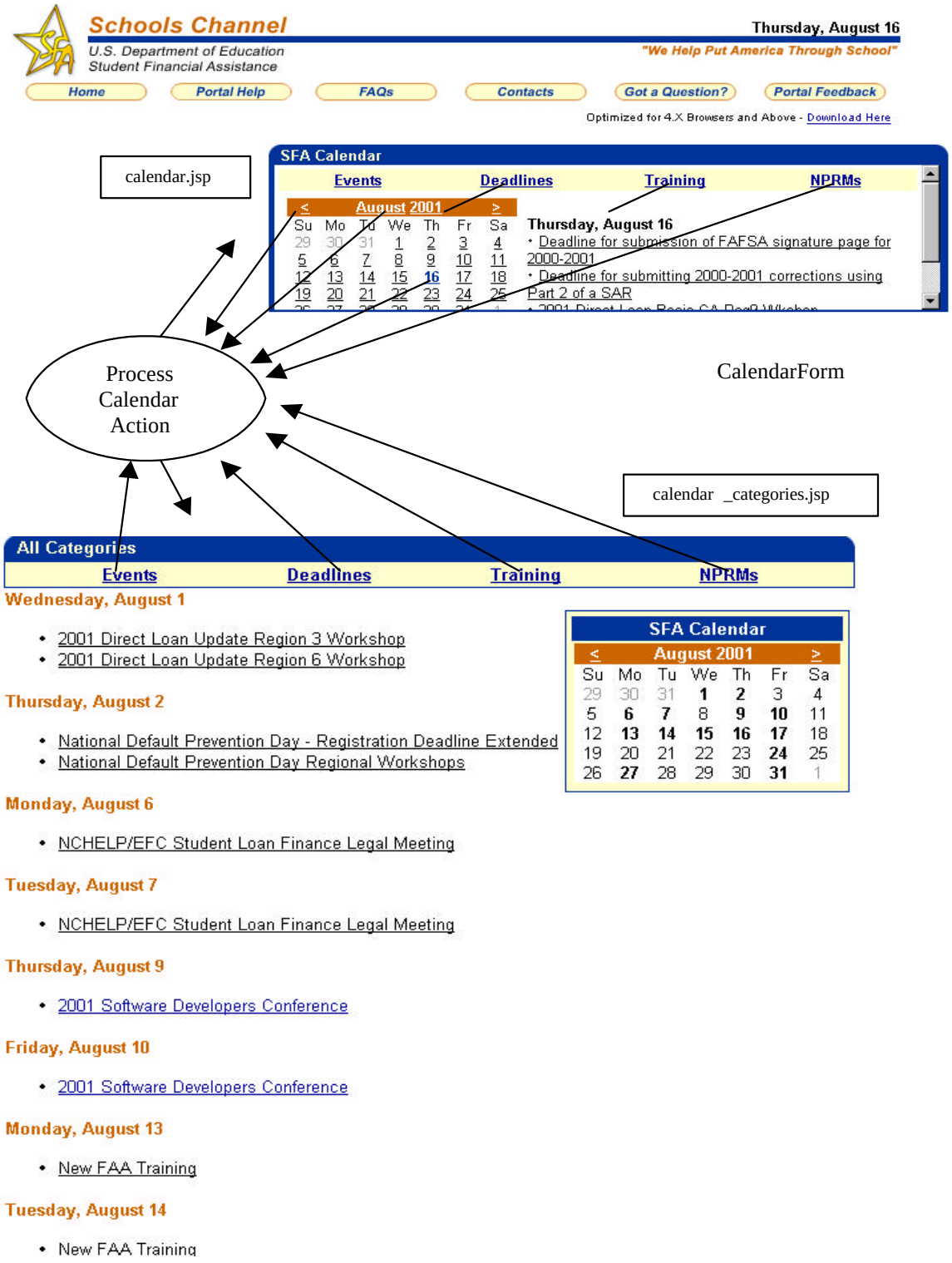


Figure 8 Calendar Portlet Navigation Schematic

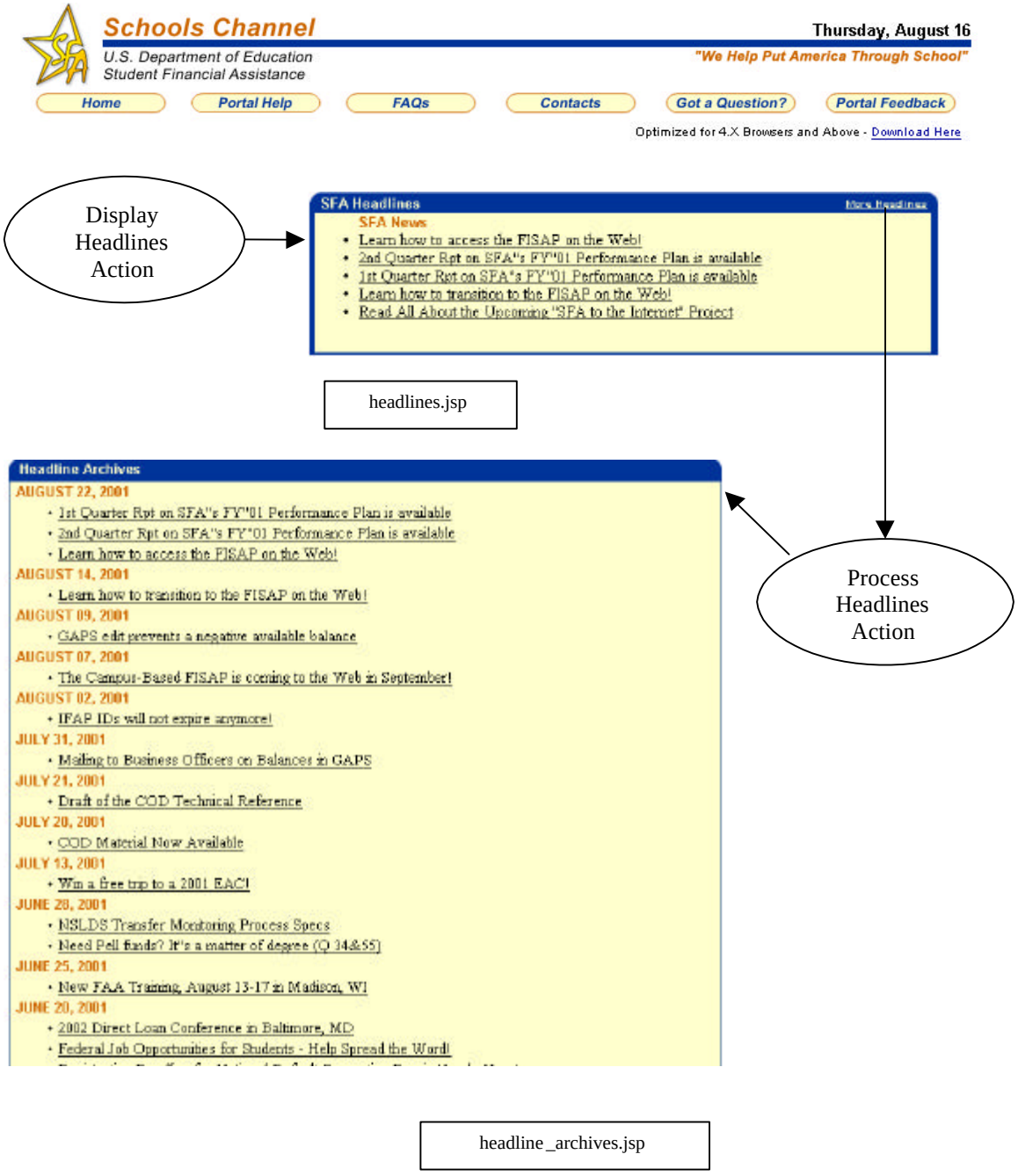


Figure 9 Headlines Portlet Navigation Schematic

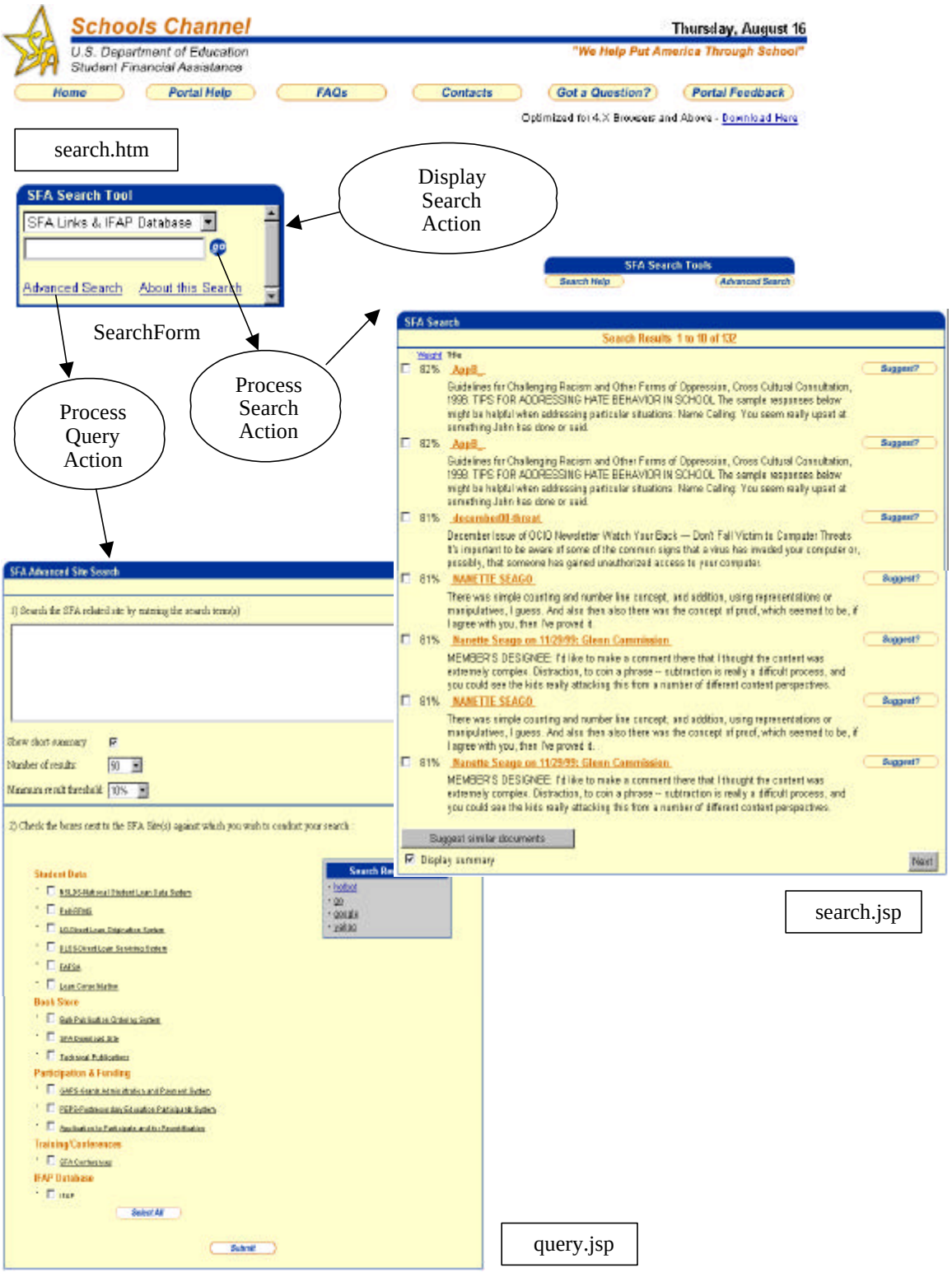


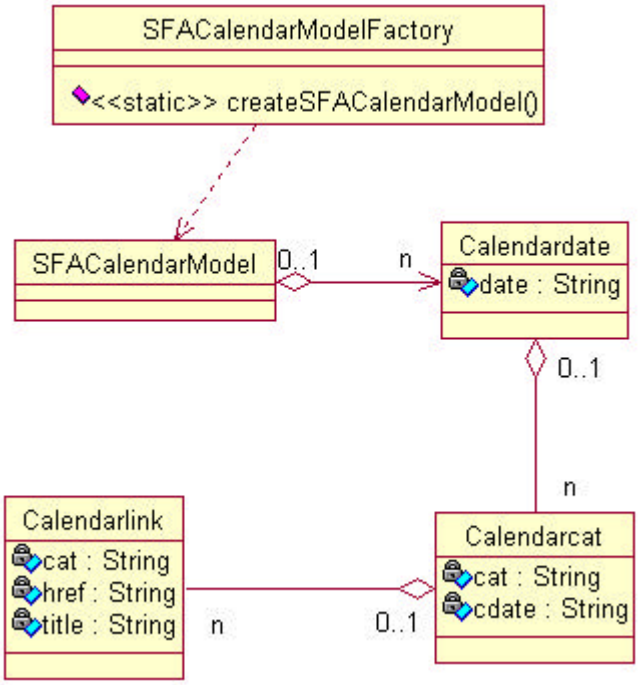
Figure 10 Search Portlet Navigation Schematic

1.5 Integration with Backend Data Sources

The key integration point with back end data sources is in the Model Bean and the Model Bean Factory.

As discussed above Model beans are used to hold information originating from back-end data- sources. The model bean's factory is responsible for creating or finding the data via create or finder methods. The model bean, as well as holding such data in single or arrayed properties, often has references to other beans, some of which are themselves model beans, forming graphs of beans.

As an example consider the relationship between SFACalendarBO, its Factory class and the other access beans which it encapsulates.



- SFACalendarBOFactory is used to create SFACalendarBO via the static createSFACalendarBO method. There is an association between SFACalendarBO and Calendardate.
- Calendardate holds a String date property. There is an association between Calendardate and Calendar (Calendar category).
- Calendarcat holds a String category name. There is an association between Calendarcat and Calendarlink.
- Calendarlink holds the category name, href and link title.

The Model beans and access beans have no part in the creation or finding process. The sole repository of knowledge regarding how to find or create a model object and its associated access beans is its factory class. Thus to modify the back-end access method, it is required is to change SFACalendarBOFactory from the one which accesses XML data sources to that which accesses relational datastores.

1.6 RCS Common Services

Phase 2 Development will use the following RCS components.

1.6.1 Exception Handling

The RCS Exception Handling component provides a framework for creating a special class of exception, SFAException. This class has methods for setting priorities not found in the java.lang.Exception class, such as the originating method name, class name, error code, and so on. The exception is created via a singleton factory object.

Using the RCS Exception handling framework in application code where new types of exceptions are needed is straightforward. These exceptions can be made to extend SFAException. However, in certain cases some exceptions already extend a base class. For example the exceptions used in the Custom Tag Handlers must be of type JspException. SFAException cannot be used in these cases.

1.6.2 Logging

The RCS Logging framework will be used to log messages via the Syslog class throughout the servlet code.

1.6.3 Persistence

The RCS persistence framework will be used in the SFA Model Factories.

1.6.4 Search

The RCS search framework will replace the current SFA CGI search function.

2 Portlet Detailed Designs

To clarify the use of Struts in building portlets, a detailed walk through of the Calendar Portlet will be presented followed by the itemized specifications of each of the other portlets.

2.1 Calendar Portlet Walkthrough

2.1.1 Default State, Category = none

In default state, the Calendar Portlet renders itself like this without a border around the heading or tabular calendar and the list view is placed on the right by the JSP HTML markup.
:-

The corresponding calendar.jsp looks like this:-

```
01: <%@ page language="java" %>
02: <%@ taglib uri="/WEB-INF/sfaportal-sfa.tld"  prefix="sfa"%>
03: <%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
04: <%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
05: <%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
06:
07: <html:html locale="false">
08: <head>
09: <title>Calendar Window Contents</title>
10: <base target="_top"></base>
11: </head>
12: <body>
13: <html:errors/>
14: <sfa:viewbean name="SFACalendarView" emit="header"/>
15: <table>
16:   <tr>
17:     <td width="60%">
18:       <sfa:viewbean name="SFACalendarView" emit="body"/>
19:     </td>
20:     <td width="40%">
21:       <logic:iterate id="calendardate" name="sfacalendarmodel" property="calendardates">
22:         <bean:write name="calendardate" property="cdate" filter="true"/><br>
23:         <logic:iterate id="calendarcat" name="calendardate" property="calendarcats">
24:            <bean:write name="calendarcat" property="cat" filter="true"/><br>
25:           <logic:iterate id="calendarlink" name="calendarcat" property="calendarlinks">
26:               <a href="<bean:write name="calendarlink" property="href"
                                                                    filter="true"/>">
27:             <bean:write name="calendarlink" property="title" filter="true"/></a><br>
28:           </logic:iterate>
29:         </logic:iterate>
30:       </logic:iterate>
31:     </td>
32:   </tr>
33: </table>
34: </body>
```

Explanation:

- Line 2 refers to the sfa Custom Tag library descriptor, further discussed in **3 The sfaportal Custom Tag Library**.
- Line 13. This is for displaying error messages such as “A database connectivity error has occurred. Please contact the Administrator.”
- Line 14. The sfa view bean declaration causes the CalendarView bean to be retrieved from the HttpRequest object and its emitHeader method to be called. See discussion below for what is emitted.
- Line 18. The SFACalendarView bean’s emitBody method is called.
- Lines 21-30. These are Struts Custom Tags, which output hierarchically organized dates, categories, and calendar links.
- The rest of the JSP is HTML markup, for example the definition of table cells to position the tabular calendar and the HTML links in relation to one another.

The SFACalendarView emitHeader method emits the table of horizontal link elements, such as:-

<td width="105" align="center">
<a target="main"
href="/sfaportal/processCalendar.do?category=events&submit=+"> **Events**</td>

The parameters passed to the ProcessCalendarAction bean via CalendarForm are discussed further below in the Calendar detail design section.

2.1.2 Selected State, Category = All, Events, Deadlines, Training or NPRMS

The format depends on the category parameter used when the HTML is initially emitted. In the category = events, deadlines, training or nprms, the heading looks like this:

In the category = all case, the heading looks like this.

Also the tabular view has a border and the list view is placed on the left by the JSP HTML markup as follows.

2.1.3 Actions, Forms, Access Beans and Model Beans

When the user clicks on a link or submit button, the GET or POST request is routed via the Struts framework to the Action object associated with the request, and the associated perform method is invoked, passing the ActionForm containing the parameters passed in the form or query string. This eliminates the need to parse the HttpRequest for parameters. There is an optional validate operation, which Struts provides, that can be invoked on the form to verify the integrity of the input.

The logic in the process method usually will create or retrieve the appropriate model bean and call methods to create the view bean and insert it in the HttpRequest object to be forwarded to the JSP.

For example, in the case of the Calendar Portlet, the ProcessCalendarAction process() method is invoked.
The SFACalendar bean is retrieved from the HttpSession, and the SFACalendar's date property, together with the category parameter passed via the CalendarForm, and some other parameters are used to determine what HTML header and body script needs to be emitted. Emitting of the appropriate HTML is done with the help of the SFACalendarEmitter in collaboration with the SFADateFormatterModel bean.

2.2 Calendar Portlet

2.2.1 Overview

The calendar portlet displays a tabular calendar view and detail for the selected day. Selection of a new day or next/previous month with the current day displays the detail HTML links for the selected day. The portlet also allows selection of items: Events, Deadlines, Training and NPRMs and All-items. Selecting one of these opens a new view in the horizontal portal, displaying a tabular calendar view and items for the month. These items may be individually selected. They are static HTML links and open a new browser window to display their content.

2.2.2 Navigation

Refer to ***Figure 8 Calendar Portlet Navigation Schematic*** showing the flow from selected links and buttons to the appropriate action objects. The actions in turn receive input from the specified forms, and route to the JSPs. Static links are not shown.

The calendar portlet consists essentially of two views, the Default view rendered in-place in the portlet frameset. This is rendered by calendar.jsp.

This default view is essentially a no-categories view. Certain selected links (such as next/previous month, and specific dates) pass the parameter category=none, to the ProcessCalendarAction object. Of course, they also pass additional parameters indicating the nature of the link, such as prev, next, or date.

Other links, such as one of the categories selected in the header section (Events, Deadlines, Training, NPRMs), or the Month-Year - equivalent to all-categories, pass one of the parameter category= events | deadlines | training | nprms | all. This forwards to the calendar_categories jsp, which we will discuss below.

One last point. Selecting one of the static links such as New FAA Training, does not route via Struts since it is a conventional static HTML link.

The second view is the Categories view. This is handled by calendar_categories.jsp

The only state information maintained between and within these views is the date. For example, in the Default view, selecting the next or previous month leaves the same date in each month selected (shown highlighted). Transitioning from the default view to the Categories view by selecting one of the categories on the top header, or the month/year in the tabular calendar, displays the categories for the selected month. There is an access bean to hold this state, kept in the HttpSession, called SFACalendar. It has a single property called date.

Similar to the Default view, selecting static links operates in the normal HTML mode.

Table 2 Explaining how the CalendarForm parameters are used

Source Page	Action	Form Parameters	Destination Page
Calendar.jsp	>	category=none next=true	calendar.jsp
Calendar.jsp	<	category=none prev=true	calendar.jsp
Calendar.jsp	1, 2, 3…	category=none date=yyyy/mm/dd	calendar.jsp
Calendar.jsp	Month Year	category=all	calendar_categories.jsp
Calendar.jsp	Events	category=events	calendar_categories.jsp
Calendar.jsp	Deadlines	category=deadlines	calendar_categories.jsp
Calendar.jsp	Training	category=training	calendar_categories.jsp
Calendar.jsp	Nprms	category=nprms	calendar_categories.jsp
Calendar_categories.jsp	>	category=<current category> next=true	calendar_categories.jsp
Calendar_categories.jsp	<	category=<current category> prev=true	calendar_categories.jsp
Calendar_categories.jsp	Month Year	category=all	calendar_categories.jsp
Calendar_categories.jsp	Events	category=events	calendar_categories.jsp
Calendar_categories.jsp	Deadlines	category=deadlines	calendar_categories.jsp
Calendar_categories.jsp	Training	category=training	calendar_categories.jsp

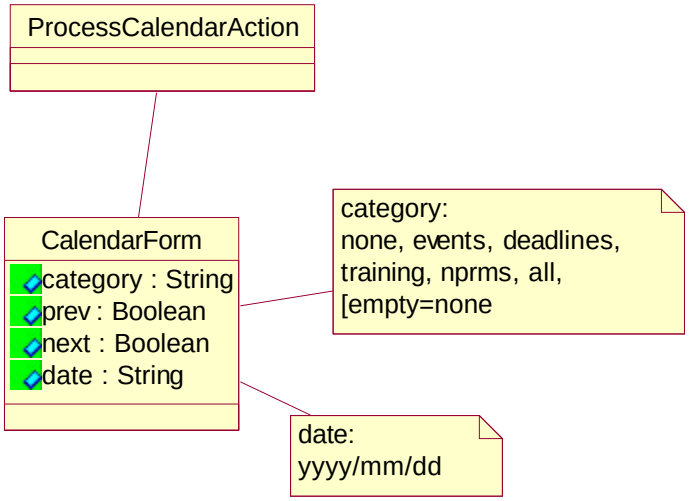
Source Page	Action	Form Parameters	Destination Page
Calendar_categories.jsp	Nprms	category=nprms	calendar_categories.jsp

JSP's and sfa tags

- calendar.jsp
  - sfa:viewbean
    - name=SFACalendarView
    - emit=header | body
- calendar_categories.jsp
  - sfa:viewbean
    - name=SFACalendarView
    - emit=header | body

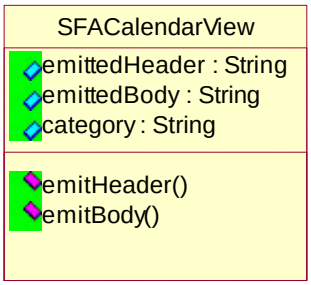
2.2.3 Actions, Forms and Form fields

- ProcessCalendarAction
  - CalendarForm

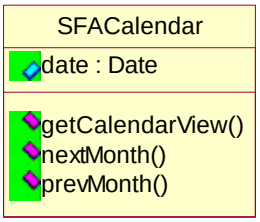


2.2.4 Access Beans and Model Beans

- SFACalendarView



  - SFACalendar



- SFACalendarEmitter
- SFADateFormatterModel
- SFACalendarBO
- SFACalendarBOFactory
- Calendardate
- Calendarcat
- Calendarlink

2.3 Horizontal Portal and Shared Header

2.3.1 Overview

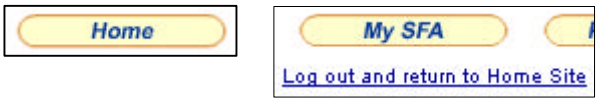
The horizontal Portal consists of a set of HTML frames and a shared header. The appearance of the header changes depending in whether the user is logged in or not.

2.3.2 Navigation

Refer to ***Figure 3 Portal Shared Header Navigation Schematic*** showing the flow from selected links and buttons to the appropriate action objects. The actions in turn receive input from the specified forms, and route to the JSPs. Static links are not shown.

2.3.3 JSP's and sfa tags

- header.jsp
  - sfa:viewbean
    - name = SFAHeaderView <Note this session scoped viewbean emits the images and link:>



2.3.4 Actions, Forms and Form fields

- DisplayHeaderAction

2.3.5 Access Beans and Model Beans

- SFAHeaderView

2.4 Login Portlet

2.4.1 Overview

The Login Portlet presents a logon view, allowing the user to enter username, password, and if registered, redisplays the portal in logged-on state. This replaces the 'Home' image in the shared header with the 'My SFA' image, also placing a link just below the image allowing the user to log off.

Attempting to log on with an unregistered username or incorrect password displays a dialog to correct these inputs.

The logon view also displays a link allowing a new user to register. This is further discussed in the Registration Portlet section below.

In logged-on state, the logon view in the portal is replaced with a view with links allowing customization of links and bookmarks. These are discussed in the Personalization Portlet section below. There is also a link allowing password change, which is part of the login portlet.

2.4.2 Navigation

Refer to ***Figure 4 Login Portlet Navigation Schematic*** showing the flow from selected links and buttons to the appropriate action objects. The actions in turn receive input from the specified forms, and route to the JSPs. Static links are not shown.

2.4.3 JSP's and sfa tags

- logon.jsp
- personalize.jsp

2.4.4 Actions, Forms and Form fields

- DisplayLogonAction
- LogonAction
  - LogonForm
    - Username
    - Password
- LogoffAction

2.4.5 Access Beans and Model Beans

- SFAUserBO
- SFAUserBOFactory

2.5 Registration Portlet

2.5.1 Overview

The Registration Portlet allows either a non logged on user to create a new registration and logon or a currently logged on user to edit their registration information.

2.5.2 Navigation

Refer to ***Figure 5 Registration Portlet Navigation Schematic*** showing the flow from selected links and buttons to the appropriate action objects. They in turn receive input from the specified forms, and route to the JSPs. Static links are not shown.

2.5.3 JSP's and sfa tags

- registration.jsp

2.5.4 Actions, Forms and Form fields

- EditRegistrationAction
  - RegistrationForm
    - Action
    - Username
    - Password
    - Password2
    - Fullname
    - EmailAddress
- SaveRegistrationAction
  - RegistrationForm
    - Action
    - Username
    - Password
    - Password2
    - Fullname
    - EmailAddress

2.5.5 Access Beans and Model Beans

- SFAUserBO
- SFAUserBOFactory

2.6 Personalization (Bookmarks) Portlet

2.6.1 Overview

The Bookmarks Personalization Portlet allows a user to edit, delete or add new personal bookmarks that will be displayed on the SFA links page when they are logged on.

2.6.2 Navigation

Refer to Figure 6 Personalization (Bookmarks) Portlet Navigation Schematic showing the flow from selected links and buttons to the appropriate action objects. They in turn receive input from the specified forms, and route to the JSPs. Static links are not shown.

2.6.3 JSP's and sfa tags

- bookmarksedit.jsp
  - sfa:linklink
    - page=/editBookmarks.do?action=Edit | /editBookmarks.do?action=Delete | /editBookmarks.do?action=Create
    - parmname=bookmarkForm
- booksmarkedit.jsp

2.6.4 Actions, Forms and Form fields

- EditBookmarkAction
  - BookmarkForm
    - Action
    - Href
    - Title

- SaveBookmarkAction
  - BookmarkForm
    - Action
    - Href
    - Title

2.6.5 Access Beans and Model Beans

- SFABookmarkBO
- SFABookmarkBOFactory

2.7 Personalization (Links) Portlet

2.7.1 Overview

The Links Personalization Portlet allows a user to customize which SFA links will be displayed on the SFA links page when they are logged on.

2.7.2 Navigation

Refer to ***Figure 7 Personalization (Links) Portlet Navigation Schematic*** showing the flow from selected links and buttons to the appropriate action objects. They in turn receive input from the specified forms, and route to the JSPs. Static links are not shown.

2.7.3 JSP's and sfa tags

- Linksedit.jsp
  - sfa:linklink
    - page=/editLinks.do?action=Delete | /editLinks.do?action=Create

2.7.4 Actions, Forms and Form fields

- EditLinkAction
  - LinkmarkForm
    - Action
    - Href
    - Title
- SaveLinkAction
  - LinkmarkForm
    - Action
    - Href
    - Title

2.7.5 Access Beans and Model Beans

- SFASystemLinksBO
- SFASystemLinksFactory

2.8 Headlines Portlet

2.8.1 Overview

The Headlines Portlet either displays the current day's SFA headlines or allow a user to request a display of all archived SFA headlines.

2.8.2 Navigation

Refer to ***Figure 9 Headlines Portlet Navigation Schematic*** showing the flow from selected links and buttons to the appropriate action objects. They in turn receive input from the specified forms, and route to the JSPs. Static links are not shown.

2.8.3 JSP's and sfa tags

- Headlines.jsp

2.8.4 Actions, Forms and Form fields

- DisplayHeadlinesAction

2.8.5 Access Beans and Model Beans

- SFAHeadlineBO
- SFAHeadlineBOFactory

2.9 Feedback Portlet

2.9.1 Overview

2.9.2 Navigation

The Feedback Portlet provides access via a static HTML link to a help page.

2.9.3 JSP's and sfa tags

There are no JSP's and sfa tags required for the Feedback Portlet.

2.9.4 Actions, Forms and Form fields

There are no actions, forms and form fields required for the Feedback Portlet.

2.9.5 View beans and presentation helper beans

There are no View beans and presentation beans required for the Feedback Portlet.

2.9.6 Access Beans and Model Beans

There are no Access or Model beans required for the Feedback Portlet.

2.10Search Portlet

2.10.1 Overview

The Search Portlet allows simple search or advanced search. The latter allows the entry of search terms and selection of those sites against which the search should be performed.

Submitting the search in either case invokes a query on the Autonomy Search engine via the RCS tag interface. All presentation and functions is implemented via the RCS tag interface.

2.10.2 Navigation

Refer to ***Figure 10 Search Portlet Navigation Schematic*** showing the flow from selected links and buttons to the appropriate action objects. They in turn receive input from the specified forms, and route to the JSPs. Static links are not shown.

2.10.3 JSP's and sfa tags

- search.htm
- searchadvanced.jsp
- searchresults.jsp
- searchsuggest.sjp

3 The SFA Portal Custom Tag Library

3.1 Introduction

A key advantage of JSP is its ability to separate presentation from implementation through the use of HTML-like tags. By reducing the use of JSP elements that embed scripting language code in the page through the use of custom tags, maintenance of JSP pages is greatly simplified, and the opportunity to reuse the Java code that provides the underlying functionality is preserved.

Custom tags are explicitly designed to add functionality to JSP pages, including the dynamic generation of page content such as HTML. In general custom tags can be used to insert text into a page, and also to implement flow of control. Attributes can be specified for custom tags, as parameters that influence their behavior. Custom tags can be empty or have bodies, which contain either nested JSP elements (including other custom tags) or tag specific content to be processed by the tag itself. Custom tags can also interact with each other, either by requesting information through the hierarchy of nested tags, or by introducing new scripting variables, which may be accessed by subsequent custom tags, as well as be the standard JSP scripting elements.

3.2 How tag libraries work

A JSP page that uses custom tags must first load the libraries containing those custom tags by means of the taglib directive. Two attributes must be specified with this directive, a URI indicating the location of the TLD file for the library, and a string specifying a page-specific XML namespace for the library's tags.
For example, see line 1 of **Figure 1 calendar.jsp tag directives**. Here the URI is `"/WEB-INF/sfaportal-sfa.tld", and the namespace prefix is "sfa".`

```
01: <%@ page language="java" %>
02: <%@ taglib uri="/WEB-INF/sfaportal-sfa.tld"  prefix="sfa"%>
03: <%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
04: <%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
05: <%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
06:
07: <html:html locale="false">
08: <head>
09: <title>Calendar Window Contents</title>
10: <base target="_top"></base>
11: </head>
12: <body>
13: <html:errors/>
14: <sfa:viewbean name="SFACalendarView" emit="header"/>
15: <table>
16:   <tr>
17:     <td width="60%">
18:       <sfa:viewbean name="SFACalendarView" emit="body"/>
19:     </td>
```

Figure 11 calendar.jsp tag directives

The tag library descriptor (TLD) file itself is an XML document. See **Figure 2 sfaportal-sfa.tld**.

As in line 10, we see there is one <tag> element for each custom tag defined in the library. The tags defined in this case, is that for a viewbean.

Line 12 specifies the tagclass. In this case it is `sfa.jsp.tags.`BeanTag, which is the Java class used to process the tag.

In line 13, the body content is designated EMPTY. XML attributes are used to specify information for the viewbean, and thus the XML element body is not required.

```
01: <?xml version="1.0" encoding="ISO-8859-1" ?>
02: <!DOCTYPE taglib
03: PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN"
"http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">
04: <taglib>
05: <tlibversion>1.0</tlibversion>
06: <jspversion>1.1</jspversion>
07: <shortname>sfa</shortname>
08: <info>SFA Portlet Tag Library.</info>
09:
10: <tag>
11: <name>viewbean</name>
12: <tagclass>sfa.jsp.tags.BeanTag</tagclass>
13: <bodycontent>EMPTY</bodycontent>
14: <info>Portlet Viewbean Tag</info>
15: <attribute>
16:   <name>name</name>
17:   <required>true</required>
18:   <rtexprvalue>false</rtexprvalue>
19: </attribute>
20: <attribute>
21:   <name>emit</name>
22:   <required>false</required>
23:   <rtexprvalue>false</rtexprvalue>
24: </attribute>
25: </tag>
26: </taglib>
```

Figure 12 sfaportal-sfa.tld

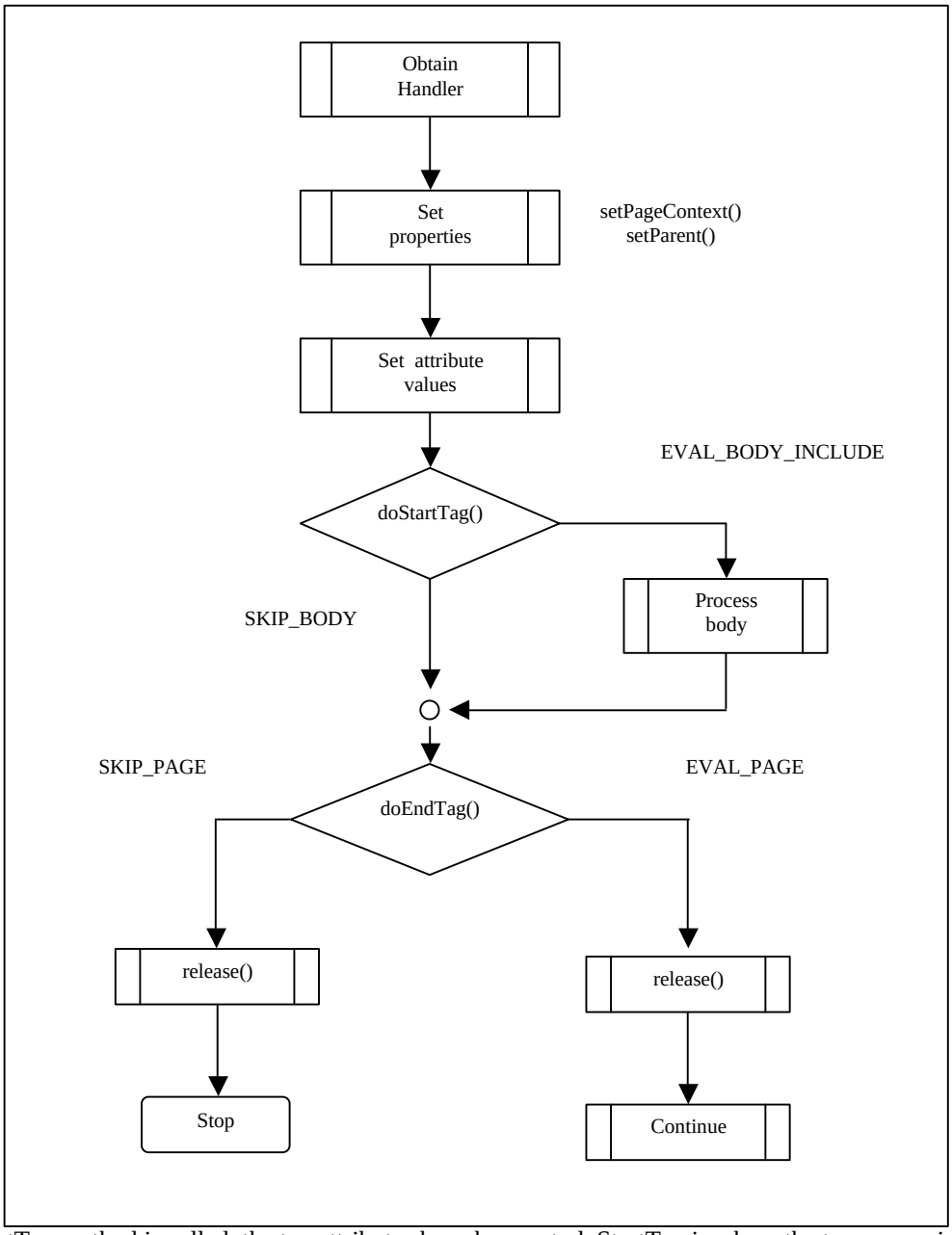
The viewbean has a mandatory **name** attribute specified in line 16 – this is why it is marked as **required** in the tld.

Depending on the name supplied, there are a set of additional attributes which specify the viewbean tag function. These are not validated by the XML parser, as is the name attribute, but need to be checked by the classes processing those viewbeans. For example, for the viewbean where name=SFACalendarView the emit attribute is not mandatory, but is checked for by the SFACalendarViewbeanProcessor Class, as described below.

3.3 Tag Handlers

Custom tags are implemented via Java objects. The tagclass element is used to specify the class that implements the handler for the tag, fully qualified with its package name. **sfa.jsp.tags.ViewbeanTag** implements the SFA custom viewbean tags and extends the javax.servlet.jsp.tagext.TagSupport class.

The life cycle of a tag handler implementing the Tag interface is shown in **Figure 3** [ref 4.]. By the time the



doStartTag method is called, the tag attributes have been set. doStartTag is where the tag processing takes place. The ViewbeanTag class's doStartTag method returns SKIP_BODY, since the tag body is empty and does not require to be processed. ViewbeanTag has access to the jsp PageContext, and is therefore able to access the important properties required to process JSPs including the request, response and session objects.

Figure 13 Life cycle of handlers implementing the tag interface

The Viewbean doStartTag method creates an instance of a class implementing the TagViewbeanProcessor interface using the abstract factory design pattern. The TagViewbeanProcessor itself is an example of the Command design

pattern. It's execute method implements the work required by the bean. For example, the SFACalendarViewbeanProcessor execute method calls emitHeader() or emitBody() on the SFACalendarViewbean itself.

The use of the abstract factory together with the command design pattern allow additional viewbean processors to be rapidly implemented. **Figure 4** shows the sequence diagram of ViewbeanTag interacting with TagViewbeanProcessorFactory, and executing an instance of a ViewBeanProcessor command.

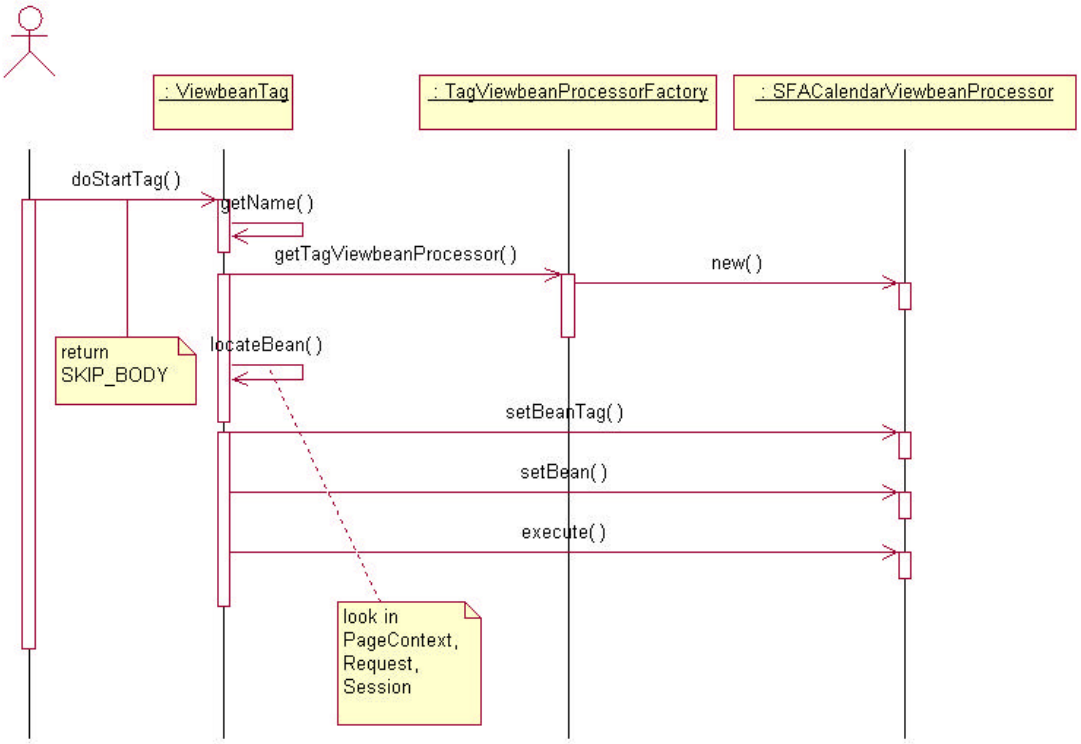


Figure 14 Tag Viewbean Processor Sequence Diagram

4 Bibliography

1. The official Struts home page: http://jakarta.apache.org/struts
2. The official Struts user guide: http://jakarta.apache.org/struts/userGuide/index.html
3. Kyle Brown, ***Apache Struts and VisualAge for Java, Part 1: Building Web-based Applications using Apache Struts (on VisualAge Developer Domain):*** http://www7.software.ibm.com/vad.nsf/data/document2558?OpenDocument&p=1&BCT=1&Footer=1
4. Kyle Brown, ***Apache Struts and VisualAge for Java, Part 2: Using Struts in VisualAge for Java 3.5.2 and 3.5.3:*** http://www7.software.ibm.com/vad.nsf/Data/Document2557?OpenDocument&SubMast=1
5. Duane Fields, Web Development with Java Server Pages. Manning Publications - 2000